Complexité de l'algorithme de l'opacité dans les systèmes Workflows centrés sur les documents

Mohamadou Lamine Diouf, Sophie Pinchinat

Complexité de l'algorithme de l'opacité

dans les systèmes Workflows centrés sur les documents

Mohamadou Lamine Diouf[(a)] et Sophie Pinchinat[(b)]

(a) Université Cheikh Anta Diop (UCAD), Dakar, Sénégal,
(b) Université de Rennes 1, Rennes, France.
mohamadoulamine@gmail.com, sophie.pinchinat@irisa.fr

RÉSUMÉ. Une propriété d'un objet est dit opaque pour un observateur si celui-ci ne peut déduire que la propriété est satisfaite sur la base de l'observation qu'il a de cet objet. Supposons qu'un certain de nombre de propriétés (appelées secrets) soient attachées à chaque intervenant d'un système, nous dirons alors que le systéme lui-même est opaque si chaque secret d'un observateur lui est opaque: il ne peut percer aucun des secrets qui lui ont été attachés.
Dans ce papier, on se propose d'étudier la complexité du problème de l'opacité des artefacts d'un système à flots de tâches(système workflow). Nous présentons une formalisation optimale du problème de l'opacité dans ces systèmes workflows. Nous étudions ensuite la complexité de l'algorithme de l'opacité pour ces systèmes.

ABSTRACT. A property (of an object) is opaque to an observer when he or she cannot deduce the property from its set of observations. If each observer is attached to a given set of properties (the so-called secrets), then the system is said to be opaque if each secret is opaque to the corresponding observer. We study in this paper, the complexity of opacity algorithm in data-centric workflows systems. We show that the complexity of this algorithm is EXPTIME-complete. Using the reduction problem, whe show that we can reduce the complexity of opacity problem to wellknow problem, the intersection of nonemptyness problem of Tree automata in polynomial time.

MOTS-CLÉS : Complexité,Opacité, système à flots de tâches, artefact, documents structurés.

KEYWORDS : Complexity, Opacity, data-centric workflow systems, artifact, structured documents.

1. Introduction

La sécurité dans les systèmes à flots de tâches et les services web occupent une place de plus en plus importante dans le dévelopement d'application, car manipulant le plus souvent des informations confidentielles. Quelques exemples de ces processus métiers qu'on peut citer sont les bases de données médicales, les systémes d'administration electronique, les systémes de commerce electronique. Dans ce contexte, le développement de techniques assurant la confidentialité des informations traitées est primordial. Afin d'étudier de tels algorithmes, la propriété de sécurité centrale qui nous intéresse ici est l'opacité, pour laquelle on étudie sa complexité dans ce papier. C'est une propriété qui caractérise l'absence de flux d'information confidentielle d'un système vers un observateur de ce système qui peut être un service web en l'occurence.
Les systèmes considérés ici sont des systèmes distribués asynchrones dans lesquels la coordination des activités s'effectue par la transmission de documents structurés (à la XML) combinant structure logique, et données. Ces systèmes sont aussi appelés système à flots de tâches centrés sur les artefacts car ils mettent plus l'accent sur les documents échangés, appelés artefacts que sur l'enchainement des procédures ou des tâches. Les artefacts sont des documents qui combinent à la fois des données et des procédés. Ces procédés peuvent être des appels de services web comme dans le modèle ActiveXML [11].
Pour qu'il y'ait communication entre le système workflows centrés sur les artefacts et les services web, il faudrait publier des informations telle qu'une adresse de messagerie, la description du service web, $\cdots$. Il faudrait également envoyer de l'information extraite du système workflow vers le service web pour plusieurs raisons.
D'une part nous voulons éviter de surcharger un service par des informations qui ne lui sont pas utiles à la réalisation de la tâche qui lui incombe. D'autre part pour des raisons de confidentialité, et c'est ce point qui nous intéresse ici, certaines informations sensibles ne doivent pas être connues de tous. Et on doit pour cela pouvoir garantir qu'il n'y a pas de fuite d'information. Cette situation nécessite le développement d'algorithmes capables de détecter ces failles de sécurité.
La notion d'opacité introduite pour la première fois dans [12] est une formalisation de l'abilité du système à garder secrète certaines informations vis à vis de l'observateur. Elle a été longtemps étudié dans le contexte des systèmes à événements discrets. Elle a servi à modéliser des problèmes de sécurité et/ou de confidentialité pour des systèmes informatiques ou des protocoles, voir e.g. [3, 9]. Ce problème a été étudié dans le contexte des systèmes à flots de tâches centrés sur les artefacts pour la première fois dans [1].
L'hypothèse sous-jacente à la notion d'opacité est que l'observateur a une parfaite connaissance du système. Ceci pour des raisons de sûreté. On considére que l'artefact est donné par une suite de documents conformes à une grammaire d'arbres modélisant le système centré sur les données, gardant secret certaines informations confidentielles contre un service. On supposera aussi que l'observateur a une vue partielle de l'artefact. Cette vue partielle est donnée par une fonction d'observation définie de l'ensemble des documents vers un sous ensemble de documents observables par effacement de sous-documents non visibles appelé sorte.
Considérant une propriété P du système, le secret S consiste à l'ensemble des documents qui vérifient P. Et une propriété P est dit opaque pour le système si chaque fois qu'un document d satisfaisant P est observable, alors il existe un autre document d' qui aura la même observation du point de vue de l'observateur et qui ne satisfait pas P. Et donc il ne

pourra pas déduire que le document d satisfait la propriété P.

Dans ce papier, nous commencons par introduire la notion de conformité d'un artefact en définissant tout d'abord la notion d'artefact dans la même section. Cette notion de conformité est toujours définie par rapport à une grammaire. Cette dernière est donnée dans la même section. La section 3, qui suit s'interesse à la structure logique d'un artefact. Dans la section 4, on y définit la notion d'information confidentielle et on donne un modèle d'observateur du système. Ensuite la notion d'opacité y est introduite. La section 5 est réservée à l'opacité dans les systèmes workflows. La complexité de l'algorithme de vérification y est étudiée. On termine par une conclusion à la section 6.

2. Conformité d'un artefact

Un artefact est un ensemble de documents constituant le dossier pour un client d'un certain service. Ces documents ont une structure d'arbre. Les informations attachées aux noeuds d'un arbre sont données par des paires attributs/valeurs qui permettent de collecter les informations pertinentes utiles pour le traitement du dossier. Néanmoins comme on aura besoin de travailler avec un ensemble fini d'étiquettes on suppose donnée une abstraction finie de ces informations sous la forme d'un alphabet Ω utilisé pour étiqueter les différents noeuds des documents. La structure arborescente reflète l'organisation logique du document : $d = \omega(d_1, \ldots, d_n)$ signifie que d est un document portant une information ω et dont $d_1, \ldots, d_n$ sont les constituants (les sous dossiers). Comme une forêt $f = d_1 \ldots d_n$ peut-être encodée par l'expression $\sharp(d_1, \ldots, d_n)$ et quitte à rajouter le symbole $\sharp$ à l'ensemble Ω, nous pourrons supposer qu'un artefact est également donné par un arbre.

Les arbres construits sur un alphabet Ω, constitué d'un ensemble fini d'éléments sont définis inductivement comme suit :

$$t \ := \omega(t_1, \ldots, t_n) \qquad \omega \in \Omega$$

ω est l'étiquette attachée à la racine de l'arbre et $t_1, \ldots, t_n$ est la liste des sous arbres se trouvant sous le noeud racine. Cette liste peut-être vide ($n = 0$) auquel cas l'arbre $t = \omega(\,)$ est réduit à son étiquette. On pourra écrire de façon abrégée $t = \omega$.

Dans cette section nous nous intéressons à caractériser la conformité d'un arbre selon une grammaire (d'arbres). Cela permet de fixer la structure logique du document, c'est-à-dire la façon dont ses différents constituants sont organisés. La notion de grammaire considérée ici correspond aux automates d'arbres (et de forêts) introduits par Murata et al [2]. Il s'agit d'une définition assez générale dans laquelle l'arité d'un noeud n'est pas fixe, comme cela est le cas pour une grammaire ordinaire. Ceci est utile lorsque parmi des constituants d'un sous dossier on peut trouver une liste d'éléments d'une certaine catégorie (par exemple la liste des ouvrages d'une bibliothèque) et dont le nombre n'est par conséquent pas déterminé, ou lorsque certains éléments sont optionnels. Le plus simple est alors d'utiliser une expression régulière pour décrire ce qu'on est censé y trouver par exemple la règle

$$A \ \rightarrow \ \omega_1\langle B^*(A + \varepsilon)\rangle + \omega_2\langle\varepsilon\rangle$$

s'interprète de la manière suivante. A et B sont des sortes (aussi appelées catégories syntaxiques). L'ensemble des sortes est fini et sert à cataloguer les différents constituants

d'un dossier en un nombre (donc fini) de catégories, par exemple ceci est un RIB, cet autre élément est une adresse ou un dossier de réservation d'hôtel. Comme dit plus haut les étiquettes $\omega \in \Omega$ sont des informations extraites des données se trouvant en racine du dossier (afin de travailler ici aussi avec un ensemble fini de possibilités). La règle précédente peut se lire alors comme suit. Un arbre est bien formé et est de la sorte A si l'information extraite à sa racine est ω_1 et il possède comme fils une suite d'éléments de sorte B dont le nombre n'est pas fixé et possiblement un élément de sorte A, ou bien l'information extraite à sa racine est ω_2 et dans ce cas il ne doit avoir aucun successeur.

Avant de définir formellement les grammaires nous introduisons la notion plus générale d'algèbre qui est mathématiquement plus simple et donc plus adaptée pour énoncer et prouver des résultats techniques.
Une algèbre est un mécanisme permettant d'extraire (ou de synthétiser) des informations à partir du document par induction sur sa structure. Exécuter une requête sur le document pourra ainsi se faire en évaluant le document selon une algèbre. Si l'information ainsi extraite est une valeur de vérité (la valeur retournée par l'évaluation est booléenne) alors l'algèbre peut être vue comme une propriété des documents. En particulier un secret, vue comme une propriété des artefacts, sera identifié à une algèbre particulière.

Définition 2.1 *Une Ω-algèbre de support D est une application $\mathcal{A} : \Omega \times D^* \to D$. On note $\omega^{\mathcal{A}} : D^* \to D$, appelée **fonction d'interprétation** de $\omega \in \Omega$, l'application définie par : $\omega^{\mathcal{A}}(x_1, \ldots, x_n) = \mathcal{A}(\omega, x_1, \ldots, x_n)$. L'ensemble $T(\Omega)$ des arbres étiquetés sur Ω est le plus petit ensemble tel que*

$$(t_1, \ldots t_n \in T(\Omega) \quad \wedge \quad \omega \in \Omega) \Longrightarrow \omega(t_1, \ldots, t_n) \in T(\Omega)$$

La valeur $t^{\mathcal{A}}$ d'un arbre $t \in T(\Omega)$ selon l'algèbre $\mathcal{A}$ est définie inductivement par :

$$t = \omega(t_1, \ldots, t_n) \Longrightarrow t^{\mathcal{A}} = \omega^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}})$$

Si $\mathcal{A}$ est une algèbre et $x \in D$ un élément de son support on pose

$$L(\mathcal{A}, x) = \left\{ t \in T(\Omega) \;\middle|\; t^{\mathcal{A}} = x \right\}$$

REMARQUE. — Notons $\mathcal{A}_{\omega,x} = \left(\omega^{\mathcal{A}}\right)^{-1} = \{u \in D^* \mid \mathcal{A}(\omega, u) = x\}$ pour $\omega \in \Omega$ et $x \in D$. Pour un symbole ω donné ces ensembles sont donc disjoints : $x \neq y \Longrightarrow \mathcal{A}_{\omega,x} \cap \mathcal{A}_{\omega,y} = \emptyset$. On note $\mathcal{A}_\omega = \bigcup_{x \in D} \mathcal{A}_{\omega,x}$ leur union. De façon similaire nous associons à $x \in D$ l'ensemble $\mathcal{A}_x = \bigcup_{\omega \in \Omega} \mathcal{A}_{\omega,x}$.

Une algèbre est non déterministe si le résultat de l'évaluation peut produire différentes valeurs. C'est-à-dire qu'il s'agit d'une fonction $\mathcal{A} : \Omega \times D^* \to \wp(D)$. On peut alors de façon classique se ramener au cas déterministe en passant aux parties c'est-à-dire en remplaçant le support D de l'algèbre par l'ensemble $\wp(D)$ des parties de D.

REMARQUE. — **Algèbre non-déterministe** — Une application $\mathcal{A} : \Omega \times D^* \to \wp(D)$, appelée algèbre non-déterministe, est assimilée à l'algèbre $\mathcal{A}^d : \Omega \times \wp(D)^* \to \wp(D)$ définie sur $\wp(D)$ par

$$\mathcal{A}^d(\omega, X_1 \cdots X_n) = \bigcup \{\mathcal{A}(\omega, x_1 \cdots x_n) \mid \forall i \in \{1, \ldots n\} \quad x_i \in X_i\}$$

Le langage d'une algèbre non déterministe est ainsi donné par $L(\mathcal{A}, x) = L(\mathcal{A}^d, x)$, c'est-à-dire $L(\mathcal{A}, x) = \left\{ t \in T(\Omega) \;\middle|\; x \in t^{\mathcal{A}} \right\}$.

La première propriété d'un artefact qu'il convient de vérifier est sa conformité à une certaine grammaire. Comme mentionné plus haut la notion de grammaire utilisée ici s'inspire des automates d'arbres de [2].

Définition 2.2 (Grammaire d'arbres régulière et langages) *Une grammaire G est la donnée d'un triplet $(\Omega, \Xi, \mathcal{L})$ où :*

- Ω : l'ensemble des symboles étiquettant les noeuds de l'arbre.

- Ξ : un ensemble fini de sortes ou types.

- $\mathcal{L} \subseteq \Xi^*$: un langage régulier associé à chaque symbole $\omega \in \Omega$ et sorte $A \in \Xi$ sur l'ensemble des sortes Ξ, noté $\mathcal{L}_{\omega,A}$

Elle sera présentée syntaxiquement sous la forme de règle de la forme $A \to \omega\langle E\rangle$, où E est une expression régulière sur Ω^* telle que $L(E) = \mathcal{L}_{\omega,A}$. Comme exemple $\mathcal{L}_{\omega,A} = B^* + C$ exprime le fait qu'un noeud ω sera de sorte A si ses successeurs immédiats sont soit une liste (éventuellement vide) de noeuds de sorte B soit un unique noeud de sorte C. Notons aussi que la remarque de tout à l'heure s'applique, ie que ces noeuds ont une arité variable : l'étiquette d'un noeud ne caractérise pas le nombre de ses successeurs.

Exemple 2.3 *Un exemple de grammaire $G = (\Omega, \Xi, \mathcal{L})$ présentée sous forme de productions avec $\Xi = \{A, B, C, D\}$ et $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6, \omega_7, \omega_8\}$:*

$$\mathcal{L} = \begin{cases} A & \to & \omega_1\langle BC\rangle + \omega_2\langle\varepsilon\rangle \\ B & \to & \omega_3\langle BD\rangle + \omega_4\langle\varepsilon\rangle \\ C & \to & \omega_5\langle CA\rangle + \omega_6\langle CCB\rangle + \omega_7\langle D\rangle \\ D & \to & \omega_8\langle AB\rangle \end{cases}$$

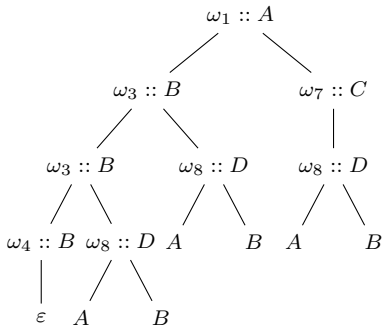


Figure 2 : Un arbre conforme à la grammaire de l'exemple 2.3

Comme dans le cas des algèbres, une grammaire non-déterministe peut être transformée en une grammaire déterministe équivalente (c'est-à-dire ayant le même langage).

Définition 2.4 (Déterminisation d'une grammaire) *La déterminisation d'une grammaire $G = (\Omega, \Xi, \mathcal{L})$ est la grammaire déterministe $G^d = (\Omega, \wp(\Xi) \setminus \{\emptyset\}, \mathcal{L}^d)$ pour laquelle $\mathcal{L}^d_{\omega,X} = \bigcup_{s\in X} \mathcal{L}_{\omega,s}$.*

La remarque suivante montre que les grammaires ne sont rien d'autres que les algèbres régulières, c'est-à-dire les algèbres dont les langages $\mathcal{A}_{\omega,x}$ sont réguliers. Par défaut une algèbre est déterministe alors qu'une grammaire est par défaut non-déterministe.

REMARQUE. — **Grammaires vs algèbres**
– Une grammaire G peut-être assimilée à l'algèbre non-déterministe $\mathcal{A}$ de domaine Ξ telle que $\mathcal{A}(\omega, u) = \{s \in \Xi \mid u \in \mathcal{L}_{\omega,s}\}$ pour $\omega \in \Omega$, $s \in \Xi$ et $u \in \Xi^*$, et donc $\mathcal{A}_{\omega,s} = \mathcal{L}_{\omega,s}$. Une grammaire déterministe G peut être assimilée à l'algèbre de domaine $D = \Xi \cup \{\top\}$ où $\top$ est un symbole supplémentaire ($\top \notin \Xi$) et

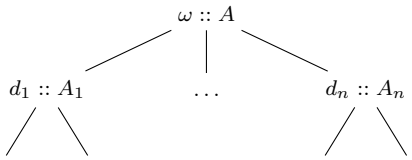
$$\omega^G(s_1, \ldots, s_n) \quad = \quad \begin{cases} s & \textbf{si} \quad s_1 \cdots s_n \in \mathcal{L}_{\omega,s} \\ \top & \textbf{sinon} \end{cases}$$

Les opérations respectives de déterminisation des grammaires et des algèbres se correspondent, c'est-à-dire que G^d décrite dans la définition 2.4 est la déterminisation de G vue comme algèbre non-déterministe : l'élément additionnel $\top$ correspond à la partie vide qu'on avait pris soin de ne pas introduire dans la définition de G^d. La correspondance entre une grammaire déterministe et l'algèbre qui lui est associée est donnée par l'identité $L(G, s) = \left\{t \in T(\Omega) \;\middle|\; t^G = s\right\}$ qui découle immédiatement des définitions 2.1 et 2.2. Un arbre est ainsi conforme à la grammaire (ou bien formé) si et seulement si $t^G \neq \top$. Remarquons que comme $\mathcal{L}_{\omega,s} \subseteq \Xi^*$ et $\top \notin \Xi$ il vient que $t = \omega(t_1, \ldots, t_n)$ est non conforme (i.e., $\omega^G(t_1^G, \ldots, t_n^G) = \top$) dès qu'un de ses sous-arbres est non conforme.

REMARQUE. — **Langage d'une grammaire** – Dans la suite nous ferons l'hypothèse suivante sur les grammaires. L'ensemble des sortes contient un symbole spécifique $\mathbf{ax} \in \Xi$, appelé **axiome** de la grammaire, pour lequel il est associé une unique production $\mathbf{ax} \to \sharp\langle E_0 \rangle$ telle que le symbole $\sharp \in \Omega$ (appelé racine) n'apparait dans aucune des autres productions de la grammaire. $t^G = s$ pour $t \in T(\Omega \backslash \{\sharp\})$ et $s \in \Xi \backslash \{\top\}$ signifie que t est bien formé de sorte s, et on pose $L(G) = \{t_1 \ldots t_n \in T(\Omega \backslash \{\sharp\})^*, t_1^G, \ldots, t_n^G \in L(E_0)\}$ le langage de la grammaire G, i.e., $t_1 \ldots t_n \in L(G)$ si, et seulement si $(\sharp(t_1, \ldots, t_n))^G = \mathbf{ax}$.
En utilisant les résultats classiques sur les langages reconnaissable d'arbres [2], [6] on peut montrer que les langages des grammaires sont clos de manière effective par les opérations booléennes. La définition suivante introduit la relation de conformité.

Définition 2.5 (Conformité) *Un document d, représenté comme un arbre sera dit conforme à une grammaire G et est de sorte A, noté $G \vdash d :: A$ si chacun de ses successeurs d_i est de sorte A_i et la suite $(A_n)_n$ appartient à $\mathcal{L}_{\omega,A}$. L'ensemble des documents conformes à la grammaire est défini inductivement comme le plus petit ensemble tel que :*

$$\begin{array}{c} \left(\forall i \in \{1, \ldots, n\} \;\; G \vdash d_i :: A_i \;\; \wedge \;\; A_1 \cdots A_n \in \mathcal{L}_{\omega,A}\right) \\ \Rightarrow \;\; G \vdash \omega(d_1, \ldots, d_n) :: A \end{array}$$



$$(\omega, A) \longmapsto \mathcal{L}_{\omega,A} \in Rat(\Xi^*)$$

que l'on note $L(G) = \bigcup_{A \in \Xi} L(G, A)$ *où* $L(G, A) = \{t \mid G \vdash t :: A\}$. *L'ensemble* $L(G)$ *est un langage d'arbre, définie comme une partie de* $T(\Omega)$.

La vérification de la conformité d'un arbre peut-être implémentée par un automate à pile. Cet automate à pile sera déterministe si $s \neq s' \Rightarrow \mathcal{L}_{\omega,s} \cap \mathcal{L}_{\omega,s'} = \emptyset$, c'est-à-dire que la sorte d'un noeud est caractérisée par son étiquette et la sorte de ses successeurs. Lorsque cette propriété est vérifiée la grammaire est dite *déterministe*.
Dans la pratique, on considèrera parfois des grammaires d'arbres locales dans lesquelles l'ensemble des sortes coïncide avec l'ensemble des symboles, car étant plus facile à manipuler.

Définition 2.6 (Grammaire locale) *Une grammaire est dite locale si son ensemble de sortes* Ξ *coïncide avec son ensemble de symboles* Ω *et* $\mathcal{L}_{\omega,s} = \emptyset$ *si* $\omega \neq s$. *Les productions d'une grammaire locale sont donc de la forme* $A \to A\langle E \rangle$ *où* A *est un symbole de la grammaire et* E *un ensemble régulier sur les sortes. La notation devient alors redondante, on écrira par conséquent ces productions sous la forme abrégée* $A \to E$.

Exemple 2.7 *Considerons la grammaire locale* $G = (\Omega, \Xi, F, \mathcal{L})$, *pour laquelle* $\Omega = \Xi$ *et pour tout* A *et* ω, $\mathcal{L}_{\omega,A} = \emptyset$ *si* $\omega \neq A$. *La grammaire est donnée par les règles suivantes :*

- $\Xi = \{A, B, C, D\}$
- $\Omega = \{A, B, C, D\}$
- $F = \{A\} \subseteq \Omega$
- $\mathcal{L} = \{A \to BC + \varepsilon, B \to BD + \varepsilon, C \to CA + CCB + D, D \to AB\}$

alors cette grammaire génère l'arbre suivant en appliquant les règles suivantes :

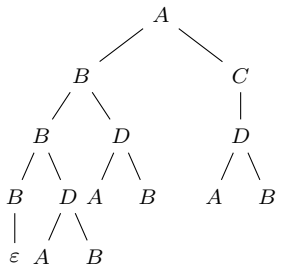


Figure 3 : Un arbre conforme à la grammaire locale G.

3. Abstraction d'un artefact

Dans la section précédente nous nous sommes intéressés à la structure logique d'un artefact (donnée par une grammaire). Nous allons maintenant nous intéresser aux données qu'il contient. On a vu que les artefacts ont un structure arborescente avec des informations attachées à chaque nœud. Ces informations attachées à un noeud correspondent à son étiquette $\omega \in \Omega$ et sont données par des paires attributs/valeurs. Les différents noeuds des documents sont étiquetés par l'alphabet Ω. Une abstraction (ou observation) consiste à supprimer toutes les informations associées à des sortes ou nœuds "non visibles" en

supprimant par la même occasion les données qui leur correspondent. L'effet de cette transformation sur la structure du document peut se décrire comme suit.

Définition 3.1 *La fonction de projection* $p_{\Xi'} : T(\Omega) \to T(\Omega)^*$ *associée à un sous-ensemble* $\Xi' \subseteq \Xi$ *(les sortes visibles) est donnée pour* $t = \omega(t_1, \ldots, t_n)$ *par :*

$$p_{\Xi'}(t) = \begin{cases} \omega(p_{\Xi'}(t_1) \ldots p_{\Xi'}(t_n)) & \textit{si} \quad t^G \in \Xi' \\ p_{\Xi'}(t_1) \cdots p_{\Xi'}(t_n) & \textit{si} \quad t^G \notin \Xi' \end{cases}$$

qu'on peut interpréter de la manière suivante :

– Si la sorte d'un arbre est visible alors sa projection est un arbre.

– Sinon c'est une suite d'arbres, appelés aussi forêts.

Les sortes visibles contiennent toujours le symbole en racine : $\sharp \in \Xi'$.

Exemple 3.2

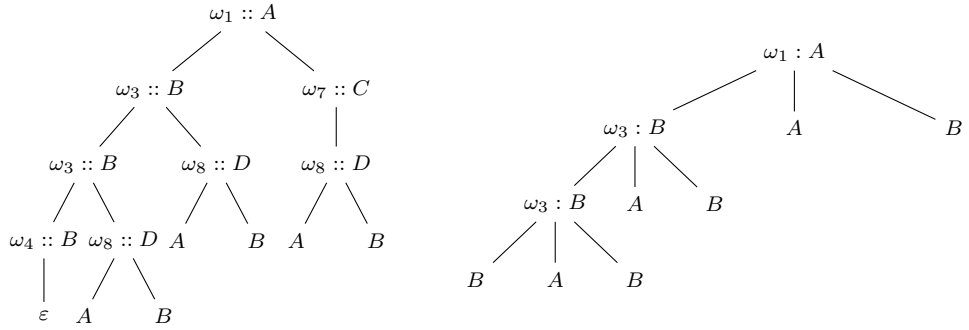


Figure 4 : L'arbre de la figure 2 et son projeté suivant le sous-ensemble de sortes visibles $\Xi' = \{A, B\}$

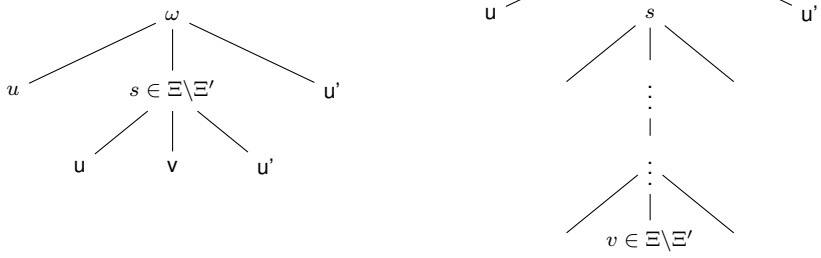
Définition 3.3 *On pose* $\mathcal{A} = G/p_{\Xi'}$ *l'algèbre non-déterministe* $\mathcal{A} : \Omega \times \Xi'^* \to \wp(\Xi')$ *définie par*

$$\mathcal{A}(\omega, v) = \{ s \in \Xi' \mid \exists u \in \mathcal{L}_{\omega,s} \quad u \vdash^* v \}$$

où $\vdash^*$ *est la fermeture réflexive et transitive de la relation*

$$\vdash \; = \{ (u \cdot s \cdot u', u \cdot v \cdot u' \mid s \in \Xi \setminus \Xi' \;\; \textit{et} \;\; v \in \mathcal{L}_s \}$$

Ainsi $\mathcal{A}^d(\omega, X_1 \cdots X_n) = \{ s \in \Xi' \mid \exists u \in \mathcal{L}_{\omega,s} \; \exists x_i \in X_i \quad u \vdash^* x_1 \cdots x_n \}$.



REMARQUE. — **Dérivation maximale** – On note $u \vdash^*_{\max} v$(v est la dérivée maximale de u) lorsque $u \vdash^* v$ (u dérive de v) et cette dérivation est maximale, c'est-à-dire qu'il n'existe pas de w dans lequel v se dérive. Si les langages $\mathcal{L}_s = \cup_\omega \mathcal{L}_{\omega,s}$ sont tous non vides —hypothèse que nous ferons par la suite— alors

$$u \vdash^*_{\max} v \iff u \vdash^* v \text{ et } v \in (\Xi')^*$$

On a le théorème fondamental suivant qui donne la relation qui existe entre le langage de la grammaire de départ et celle observée.

Théorème 3.4 *([1])* $L(G/p_{\Xi'}) = p_{\Xi'}(L(G))$.

Les sections qui suivent abordent les notions fondamentales pour traiter le problème de l'opacité.

4. Notion d'information confidentielle et modèle d'observateur

Définition 4.1 *Une propriété secrète S sur les arbres de notre système $G = (\Omega, \Xi, \mathcal{L})$ est l'ensemble des arbres tels que s'ils vérifient la propriété S, alors constituent les informations confidentielles qu'on souhaite préserver d'un observateur.*

On dit qu'un arbre t vérifie la propriété S lorsque t appartient à S. On note $t \in S$, et $t \notin S$ si l'objet ne vérifie pas la propriété.
Soit un observateur représenté par sa fonction d'observation $p_{\Xi'}$ dans laquelle Ξ' est l'ensemble des sortes visibles et $p_{\Xi'}(t)$ est l'information que l'observateur a de l'arbre t. Le but de cet observateur est d'inférer de l'information sur t, sachant que l'observateur a une parfaite connaissance du système donné par la grammaire G, il sait par ailleurs comment sa fonction d'observation $p_{\Xi'}$ est construite et il est en mesure de déterminer l'ensemble $p^{-1}_{\Xi'}(t)$ des arbres qui donnent lieu à une observation donnée. De cette façon, la connaissance qu'il a d'un arbre $t \in G$ est donnée par l'ensemble $p^{-1}_{\Xi'}(p_{\Xi'}(t))$ des arbres donnant lieu à la même observation que celui-ci. Ainsi il sera capable de déterminer qu'un arbre t vérifie la propriété S lorsque l'ensemble $p^{-1}_{\Xi'}(p_{\Xi'}(t))$ est complétement inclus dans S. Ce que traduit la formule suivante :

$$p^{-1}_{\Xi'}(p_{\Xi'}(t)) \subseteq S$$

Lorsque cette relation est vérifiée par un certain arbre, on dit qu'il y a fuite d'information, ce qui constitue une faille de sécurité dans le cas où la propriété qui a été déduite est une information sensible qu'on souhaitait garder secrète. L'élément t vérifiant la relation ci-dessus est appelé un témoin de la fuite de l'information P.

4.1. Notion d'opacité

Définition 4.2 *Une propriété secrète est opaque sur G pour un observateur si pour tout observation d'un arbre de G satisfaisant la propriété S, noté $t \models S$, il existe un autre arbre t', ayant la même observation. On peut l'exprimer formellement*

$$\forall t \in S \; \exists t' \notin S \; p_{\Xi'}(t) = p_{\Xi'}(t')$$

On peut reformuler la formule ci-dessus de façon ensembliste.

Définition 4.3 *La propriété S est opaque vis-à-vis de $p_{\Xi'}$ si et seulement si $p_{\Xi'}(S) \subseteq p_{\Xi'}(T(\Omega)\backslash S)$.*

Ces deux définitions indiquent que l'observateur possède une information partielle sur ce qui se passe réellement quand un arbre est produite par la grammaire G.
On peut introduire la relation d'équivalence θ sur les arbres pour modéliser cette notion d'information partielle de l'observateur. Sa sémantique est donnée par :
si un arbre engendré par la grammaire G vérifie S, alors il existe un autre arbre t' ne vérifiant pas S tel que $t \simeq_\theta t'$, alors l'observateur ne saurait discerner t et t' mais sait qu'au moins un arbre de la classe de $[t]_\theta$ est observable. Dans ce contexte l'observateur connaissant $[t]_\theta$, souhaiterait inférer un arbre t satisfaisant S, noté $t \models S$. En particulier il est sûr que $t \models S$ lorsque tout arbre de $[t]_\theta$ satisfait S.
REMARQUE. — On simplifie dans ce papier la notion de secret en le considérant comme propriété des arbres. Dans ce travail préliminaire on ne cherche pas à identifier une syntaxe pour exprimer les secrets sur les documents.

5. Opacité dans les systèmes workflows

Etant donné l'ensemble C des arbres reconnnus par la grammaire $G = (\Omega, \Xi, \mathcal{L})$. Soit $\Xi' \subseteq \Xi$ un sous-ensemble de sortes visibles. Les observations partielles sont données par la fonction d'observation $p_{\Xi'}$, et pour chaque observateur, une propriété des arbres représentée par la grammaire G_S dont le langage reconnu par cette grammaire est appelée secret et est notée S.
On s'interroge sur la possibilité d'un observateur d'inférer qu'un arbre du système se trouve dans le secret, à partir de la vue partielle qu'il a de cet arbre.

La figure suivante illustre les propos ci-dessus :
L'ellipse rouge décrit l'ensemble des secrets $S = L(G_S)$, le carré bleu décrit notre système, $C = L(G)$ et le grand carré noir représente l'ensemble des arbres construit sur l'alphabet Ω. La bande bleue représente les arbres pour lesquels le système n'est pas opaque. Le trait bleu horizontal représente les arbres projetés, le petit rectangle rouge sur cette bande représente L_{pbm}.
L_{pbm} décrit la projection des arbres contre-exemples, i.e des arbres pour lesquels le système n'est pas opaque. Le secret S est opaque vis-à -vis de $p_{\Xi'}$ si et seulement si L_{pbm} est vide, c'est à dire que $p_{\Xi'}(S \cap C) \leq_c p_{\Xi'}(C \setminus S)$.
Le langage $\mathcal{L}_{\omega,s}$ associé à la grammaire $\mathcal{G} = (G/p_{\Xi'})$ qui caractérise les projections des documents conformes à la grammaire G, i.e., $L(\mathcal{G}) = L(p_{\Xi'}(G))$, est un *langage algébrique*. Ces langages sont rationnels si la grammaire est *non récursive*, c'est-à-dire si on ne peut trouver de cycles dans la relation de dépendance $s > s'$ correspondant au fait que s' apparait en partie droite d'une règle associée à s. Les grammaires sont closes de manière effective par les opérations booléennes. La complémentation et donc l'union ne préservent pas la non-circularité. Néanmoins les deux opérations qui nous intéressent, à savoir l'intersection et la différence, préservent cette propriété. Le problème ci-dessus peut donc être résolu de manière effective dans ce cas. On peut construire la grammaire $p_{\Xi'}(S \cap C) \setminus p_{\Xi'}(C \setminus S)$ qui reconnait exactement les cas de violation du secret. Dans le cadre d'un outil interactif cette grammaire peut être utilisée pour fournir des contre-exemples —les témoins— lorsque l'opacité n'est pas assurée. Cette information peut

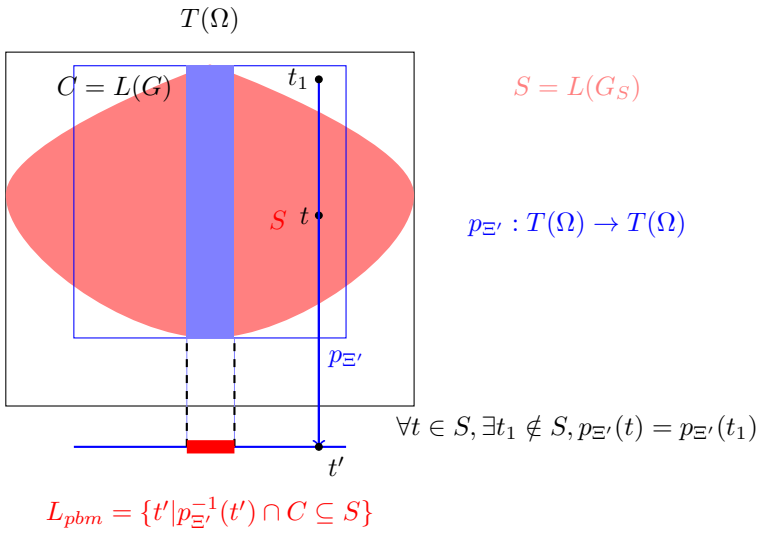


Figure 5 : Opacité dans les systèmes workflows

être utilisée par le concepteur du système pour l'aider à modifier la description du workflow en vue d'en assurer l'opacité.

Il est assez naturel de considérer comme nous l'avons fait des grammaires dans lesquelles les parties droites des productions sont des langages réguliers (et non un ensemble fini de mots) car on aura souvent besoin de considérer qu'un noeud d'un artefact puisse contenir une liste de sous documents d'une certaine sorte sans que la quantité de tels documents puissent être fixée *a priori*. En revanche on aura rarement besoin de considérer des grammaires récursives. Ainsi du point de vue des artefacts l'hypothèse de non circularité est tout à fait raisonnable. Néanmoins comme nous l'avons signalé plus haut dans le texte cette hypothèse restreindra la classe des secrets que nous pourront prendre en compte puisqu'on ne pourra pas exprimer des propriétés qui dépendent d'information contextuelles.

La solution alternative que nous considérons ci-dessous est d'interpréter les documents modulo permutations des sous arbres d'un noeud. Nous restons de cette façon dans le cadre rationnel grâce au thérème de Parikh qui nous dit que l'image commutative d'un langage algébrique est rationnel.

Nous considérons que l'ordre dans lequel les successeurs d'un noeud apparaissent n'est pas significatif. Pour prendre en compte cette contrainte on pourrait décider de se restreindre aux grammaires pour lesquelles les langages $\mathcal{L}_{\omega,A}$ sont clos par commutations (la plus petite congruence $\approx$ pour laquelle $\omega \cdot \omega' \approx \omega' \cdot \omega$). Mais ceci peut être beaucoup trop restrictif. Pour cette raison, nous allons plutôt considérer la relation de "conformité modulo permutations" définie comme suit.

Définition 5.1 (Conformité modulo permutations) *La relation de conformité modulo permutations est la plus petite relation telle que :*

$$(\forall i \in \{1, \ldots, n\} \quad G \vdash_c t_i :: s_i \quad \wedge \quad s_1 \cdots s_n \in [\mathcal{L}_{\omega,s}])$$

$$\Rightarrow \quad G \vdash_c \omega(t_1, \ldots, t_n) :: s$$

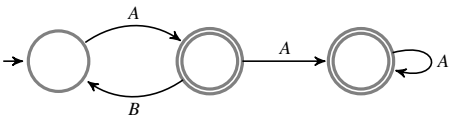
où $[L] = \{u \mid \exists v \in L \quad u \approx v\}$ *est la clôture de* L *modulo commutations. Le* ***langage modulo commutations*** *d'une grammaire est donné par :*

$$\begin{aligned} L_c(G,s) &= \{t \in T(\Omega) \mid G \vdash_c t :: s\} \\ L_c(G) &= \{t_1 \dots t_n \in T(\Omega) \mid G \vdash_c \sharp(t_1,\dots,t_n) :: \mathbf{ax}\} \end{aligned}$$

REMARQUE. — Pour que la procédure de reconnaissance soit déterministe nous supposerons, non seulement que les différents langages $\mathcal{L}_{\omega,s}$ —lorsque s varie— sont disjoints, mais que leurs clôtures commutatives le sont également : $s \neq s' \Rightarrow [\mathcal{L}_{\omega,s}] \cap [\mathcal{L}_{\omega,s'}] = \emptyset$.

Dans la mesure où la clôture par commutations d'un langage rationnel n'est généralement pas un langage rationnel, cette seconde approche est plus générale.

Exemple 5.2 *La clôture par permutation de* $\mathcal{L} = (AB)^* \cdot A^*$ *est l'ensemble des mots dont le nombre d'occurrences de* A *est supérieur ou égal au nombre d'occurrences de* B*. Ce dernier n'est pas rationnel (il possède un ensemble infini de résidus). On peut utiliser l'automate associé au langage régulier* $\mathcal{L}$



comme un générateur *de* $[\mathcal{L}]$*. Pour cela on interprète l'expression rationnelle* $(AB)^*A^*$ *dans* N^2 *en identifiant les lettres* A *et* B *avec les deux générateurs* $(1,0)$ *et* $(0,1)$ *respectivement. Au niveau de l'automate cela revient à cumuler les valeurs des étiquettes rencontrées le long d'un chemin reconnaissant de l'automate. Cet automate procure ainsi un mécanisme qui permet d'engendrer* $[\mathcal{L}] = \{(x,y) \in N^2 \mid x \geq y\}$*. Cet ensemble est associé à la formule* $F(x,y) \equiv (\exists z) \cdot x = y + z$*. Néanmoins cet automate ne peut être utilisé comme un reconnaisseur.*

REMARQUE. — La reconnaissance de documents modulo permutations est pris en compte dans le formalisme de *RelaxNG* à l'aide du pattern *interleave* [12] : il se traduit dans la syntaxe XML par le tag *interleave* qui permet de ne pas considérer l'ordre d'apparition des éléments. Malheureusement, Il est diffcile de trouver les algorithmes utilisés pour la reconnaissance de ces types de documents.

Les ensembles qui nous intéressent sont donc les images commutatives des langages rationnels, c'est-à-dire les parties rationnelles de N^k (où k est la taille de l'alphabet). Le fait que ces ensembles ne soient pas reconnaissables pourrait sembler constituer un obstacle pour la vérification de la conformité d'un document (modulo commutations). En fait il n'en est rien, bien que ce processus va devoir suivre une procédure un peu plus compliquée que dans le cas de base. Les parties rationnelles de N^k coïncident avec les parties semi-linéaires.

RAPPEL. — Une partie linéaire est l'ensemble des vecteurs de la forme $v_0 + n_1 \cdot v_1 + \dots + n_\ell \cdot v_\ell$ (une expression affine dans laquelle les vecteurs v_0 —l'origine— et $v_1, \dots v_\ell$

—les vecteurs de base— sont fixés et les variables n_i représentent des entiers arbitraires). Une partie semi-linéaire est une union finie de parties linéaires.

Ces ensembles semi-linéaires coïncident par ailleurs avec les ensembles définissables par des formules de Presburger, i.e., de la forme $\{(x_1, \cdots, x_n) \mid F(x_1, \cdots, x_n)\}$ où F est une formule de Presburger.

RAPPEL. — La logique de Presburger est le fragment de l'arithmétique dans lequel on exclut la multiplication, c'est-à-dire qu'il s'agit du calcul propositionnel avec quantification sur les entiers et avec l'addition. On peut bien sûr écrire la multiplication par une constante puisque par ex. $3x = x + x + x$. De la même manière on peut écrire des inéquations : par ex. $x \leq y$ est une abréviation de $(\exists z) \cdot \ y = x + z$.

Ces différentes correspondances sont effectives : on peut passer entre expressions régulières, ensembles semi-linéaires et formules de Presburger en utilisant des algorithmes bien établis (voir [7],[8],[10], [4]). L'intérêt de la logique de Presburger est sa décidabilité (au contraire de l'arithmétique dans son ensemble, qui est indécidable). Comme pour la décision de la logique du premier ordre, la méthode consiste à construire (par induction sur la structure de la formule) un automate qui reconnait l'ensemble des vecteurs vérifiant la formule (pour un codage particulier des vecteurs en des mots sur un certain alphabet). La validité de la formule se réduit en la non vacuité de l'automate correspondant. Nous utilisons cette construction pour vérifier comme suit la validité d'un arbre. Considérant un noeud étiqueté par un opérateur ω, on suppose inductivement que chacun de ses sous-arbes immédiats ait été jugé conforme avec une sorte déterminée, on collecte ces différentes sortes pour former un vecteur dont on vérifie qu'il satisfait la formule de Presburger associée à l'image commutative de $\mathcal{L}_\omega = \cup_{s \in \Xi} \mathcal{L}_{\omega,s}$.

Les ensembles semi-linéaires sont par ailleurs clos de manière effective par les opérations booléennes.

La définition suivante est l'adaptation de la définition 3.3 pour les grammaires à commutations près.

Définition 5.3 *Si $\Xi' \subseteq \Xi$ est un sous-ensemble de l'alphabet des sortes d'une grammaire $G = (\Omega, \Xi, \mathcal{L})$ et $s \in \Xi \setminus \Xi'$, on pose $L(G, \Xi', s) = \{u \in \Xi'^* \mid s \vdash^* u\}$. La projection de G à Ξ' est la grammaire $p_{\Xi'}(G) = (\Omega, \Xi', \mathcal{L}')$ où*

$$\mathcal{L}'_{\omega,s'} = \mathcal{L}_{\omega,s'}[[L(G, \Xi', s)]/s \ ; \ s \in \Xi \setminus \Xi']$$

c'est-à-dire qu'on substitue l'image commutative du langage algébrique $L(G, \Xi', s)$ (qui est donc un langage rationnel par le théorème de Parikh) à la variable non visible $s \in \Xi \setminus \Xi'$ dans la partie droite $\mathcal{L}_{\omega,s'}$ de la règle, dans la grammaire d'origine, associée au symbole visible $s' \in \Xi'$.

Dans la pratique nous n'introduisons pas explicitement le langage régulier $[L(G, \Xi', s)]$ mais on exhibe une expression régulière qui le caractérise à commutations près. Pour cela on utilise la proposition suivante :

Proposition 5.4 ([14]) *Si la règle pour le symbole X se factorise (modulo commutation) en $X =_c A(X) \cdot X + T$ dans laquelle la variable X n'apparait pas dans T, alors la plus petite solution de cette équation est à commutations près donnée par l'expression régulière $A(T)^* \cdot T$.*

Exemple 5.5 (voir Exemple 2.5)
La projection de la grammaire

$$G = \left(\begin{array}{l} A \quad \textbf{where} \\ \quad A \to \omega_1\langle BC\rangle \ + \ \omega_2\langle\varepsilon\rangle \\ \quad B \to \omega_3\langle BD\rangle \ + \ \omega_4\langle\varepsilon\rangle \\ \quad C \to \omega_1\langle CA\rangle \ + \ \omega_4\langle CCB\rangle \ + \ \omega_1\langle D\rangle \\ \quad D \to \omega_2\langle AB\rangle \end{array} \right)$$

sur le sous-alphabet $\Xi' = \{A, B\}$ *est donnée par*

$$p_{\{A,B\}}(G) = \left(\begin{array}{lll} A & \textbf{where} & \\ \quad A & \to & \omega_1\langle BC\rangle \ + \ \omega_2\langle\varepsilon\rangle \\ \quad B & \to & \omega_3\langle BD\rangle \ + \ \omega_4\langle\varepsilon\rangle \\ \quad C & =_c & CA + CCB + D \\ \quad D & =_c & AB \end{array} \right)$$

En utilisant $CA + CCB + D =_c (A + CB)C + D$ *on obtient* $C =_c (A + DB)^* D = (A + ABB)^* AB$ *et donc*

$$p_{\{A,B\}}(G) = \left(\begin{array}{lll} A & \textbf{where} & \\ \quad A & \to & \omega_1\langle B(A + ABB)^* AB\rangle \ + \ \omega_2\langle\varepsilon\rangle \\ \quad B & \to & \omega_3\langle BAB\rangle \ + \ \omega_4\langle\varepsilon\rangle \end{array} \right)$$

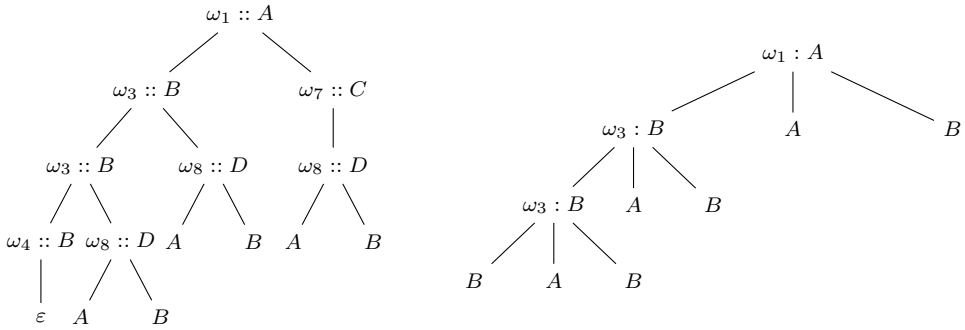


Figure 6 : A gauche, l'arbre t *conforme à la grammaire* G *:* $G \vdash_c t :: A$*. A droite, la projection de* t *sur le sous-alphabet de sortes visibles* $\Xi' = \{A, B\}$*. On vérifie que* $p_{\{A,B\}}(G) \vdash_c p_{\{A,B\}}(t) :: A$*.*

5.1. Vérification de l'opacité

Dans cette section on étudie la complexité du problème de l'opacité d'un système workflows centré sur les données.

Problème de l'opacité

Entrée : Des automates d'arbres finis déterministes A_G *et* A_S sur l'ensemble des sortes Ξ, *et* $\Xi' \subseteq \Xi$ un sous-ensemble de sortes visibles.
Sortie : Le langage $L(A_S)$ est-il opaque sur $L(A_G)$ vis-à-vis de $p_{\Xi'}$?

Théorème 5.6 *Le problème de l'opacité est EXPTIME-complet.*

Le reste de cette section est consacré à la preuve du Théorème 5.6.

REMARQUE. — Contrairement à ce que nous avons présenté jusqu'ici, le système workflow est représenté par un automate d'arbres et non par une grammaire d'arbres. On peut cependant choisir indifféremment parmi ces représentations puisque l'on peut passer d'une représentation des automates d'arbres à une représentation des grammaires d'arbres et inversement en temps polynomial. Ainsi on associera dans la suite la grammaire d'arbres G (resp. G_s) à l'automate d'arbres déterministe A_G (resp. A_S).

Démonstration 5.7 *Pour la borne supérieure on se propose de calculer la complexité de l'algorithme ci-dessous qui teste l'opacité d'un système donné par un automate d'arbres déterministe :*

Algorithme de vérification de l'opacité

Entrées :Des automates d'arbres finis déterministes A_G et A_S sur l'ensemble des sortes Ξ, et $\Xi' \subseteq \Xi$ un sous-ensemble de sortes visibles.

1. *Calculer $A_G \cap A_S$*
2. *Calculer $A_G \backslash A_S$*
3. *Calculer $p_{\Xi'}(A_G \cap A_S)$.*
4. *Calculer $p_{\Xi'}(A_G \backslash A_S)$.*
5. *Tester la vacuité de l'automate d'arbres $A = p_{\Xi'}(A_G \cap A_S) \backslash p_{\Xi'}(A_G \backslash A_S)$.*

Théorème 5.8 *L'algorithme de vérification de l'opacité est EXPTIME.*

Démonstration 5.9 (du Théorème 5.8) *On va d'abord énoncer le résultat suivant :*

Proposition 5.10 *[6] Les arbres à arité variable non ordonnées sont clos par opération booléennes.*

Sachant également que la projection d'une grammaire d'arbres régulière selon $p_{\Xi'}$ est une grammaire régulière, on peut donner la preuve du théorème 5.8 :

1. *La compléxité des calculs $A_G \backslash A_S$ et $A_G \cap A_S$ est polynomial en temps.*
2. *Le projeté de l'automates d'arbres $A_G \cap A_S$, noté $p_{\Xi'}(A_G \cap A_S)$ est un automate d'arbres modulo commutation. Le calcul de sa complexité est linéaire en la taille des symboles non-visibles $\Xi \backslash \Xi'$ donc polynomial aussi.*
3. *Idem également pour le calcul de $p_{\Xi'}(A_G \backslash A_S)$*
4. *Tester la vacuité de l'automate $p_{\Xi'}(A_G \cap A_S) \backslash p_{\Xi'}(A_G \backslash A_S)$ est équivalent au problème de l'inclusion des automates d'arbres non déterministes qui s'énonce :*

Problème de l'inclusion d'automates d'arbres :

Entrée : *Deux automates d'arbres B_1, B_2 sur l'ensemble des sortes Ξ*
Sortie : $L(B_1) \subseteq L(B_2)$?

Théorème 5.11 *([6]) Le problème de l'inclusion d'automates d'arbres non déterministes est EXPTIME.*

Pour la borne inférieure qui porte de nouveau sur le Théorème 5.6, nous proposons une réduction du problème de la vacuité pour l'intersection d'automates d'arbres, qui s'énonce comme suit :

Problème de la vacuité de l'intersection d'automates d'arbres finis :

Entrée : *un automate d'arbres déterministe* A_1 *et un automate d'arbres* A_2 *sur l'ensemble des sortes* Ξ
Sortie : $L(A_1) \cap L(A_2) = \emptyset$?

On sait que :

Théorème 5.12 *([13]) Le problème de la vacuité de l'intersection non vide d'automates d'arbres et d'automates d'arbres déterministes est EXPTIME-complet.*

Soient donc $\mathcal{A}_1$ *un automate d'arbres déterministes et* $\mathcal{A}_2$ *un automate d'arbres sur l'ensemble des sortes* Ξ*. On définit* $T(\Omega)^\sharp$ *l'ensemble des arbres construits au dessus de* $\Omega \cup \{\sharp\}$*, où* $\sharp$ *est un nouveau symbole. Si* $L \subseteq T(\Omega)$*, on lui associe le langage* $L^\sharp = \{\sharp(t) \mid t \in L\} \subseteq T(\Omega)^\sharp$*. Il est facile de voir que* $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) = \emptyset$ *ssi* $L(\mathcal{A}_1)^\sharp \subseteq L(\mathcal{A}_2)^{\sharp c}$*.*
On définit les langages $C = L(\mathcal{A}_1)^\sharp$ *et* $S = L(\mathcal{A}_2)^{\sharp c}$ *sur* Ξ*, et soit* $\Xi' = \{\sharp\} \subseteq \Xi$*, l'ensemble des sortes visibles réduit au seul symbole* $\sharp$*. Alors pour tout arbre* $t \in T(\Omega)^\sharp$*,* $p_{\Xi'}(t) = \sharp$ *; et on a*

$$L_{pbm} = \left\{\sharp \;\middle|\; p_{\Xi'}^{-1}(\sharp) \cap C \subseteq S\right\} = \left\{\sharp \;\middle|\; T(\Omega)^\sharp \cap C \subseteq S\right\}$$

Or S *est opaque sur* C *vis-à-vis de* $p_{\Xi'}$ *si et seulement si* $L_{pbm} = \emptyset$ *si et seulement si* $C \subseteq S$ *i.e.* $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) = \emptyset$*.*

6. Conclusion

Dans ce papier, nous avons présenté le formalisme utilisé pour étudier le problème de l'opacité dans le contexte des système à flots de tâches centrés sur les documents. Nous avons proposé un algorithme de décision qui peut être utilisé dans le cadre d'un outil interactif par le concepteur du système pour opérer les modifications nécessaires au rétablissement de l'opacité. Nous avons pu établir que l'algorithme de décision proposé dans ce papier est optimal.
Dans la suite, nous comptons appliquer cet algorithme dans le contexte de la protection des données personnelles et plus particulièrement dans la protection des données de santé [17]. Nous souhaitons également proposer des méthodes pour rendre le système opaque dans le cas où il ne serait pas : soit par contrôle en excluant tous les documents qui dévoile le secret et s'inspirer de [16], ou par restructuration de la grammaire en s'inspirant de techniques d'extensions conservatives de schémas XML[15].
Nous comptons également implémenter les algorithmes de décision et de contrôle de l'opacité en utilisant les bibliothèques existantes, tel que l'outil Xduce développé par Hosoya et al [13].

7. Bibliographie

[1] E. BADOUEL AND M.L. DIOUF, « Opacité des artefacts d'un système Workflow, *Revue ARIMA*, vol. 17, n° 177–196, 2014.

[2] ANNE BRUGGEMANN-KLEIN, A. MAKOTO MURATA, DERICK WOOD, « Regular Tree and Regular Hedge Languages over Unranked Alphabets », *HKUST-TCSC-2001-05*, 2001.

[3] JEREMY BRYANS, MACIEJ KOUTNY, LAURENT MAZARÉ, PETER Y. A. RYAN, « Opacity generalised to transition systems », *Int. J. Inf. Sec.*, vol. 7,n° 6, 2008 :421-435.

[4] FABRICE CHEVALIER, JÉRÉMIE CHALOPIN, « Représentation et algorithmique des ensembles semi-linéaires », *Rapport de stage, LSV - ENS Cachan*, 2001.

[5] DAVID COHN, RICHARD HULL, « Business Artifacts : A Data-centric Approach to Modeling Business Operations and Processes », *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2009.

[6] HUBERT COMON, MAX DAUCHET, RÉMI GILLERON, FLORENT JACQUEMARD, DENIS LUGIEZ, CHRISTOF LÖDING, SOPHIE TISON, MARC TOMMASI, « Tree Automata Techniques and Applications », `http://tata.gforge.inria.fr`, 2008.

[7] SEYMOUR GINSBURGH, « The Mathematical Theory of Context-Free Languages », *Mc Graw-Hill*, 1966.

[8] SEYMOUR GINSBURGH, E.H. SPANIER, « Semigroups, Presburger formulas, and languages », *Pacific Journal of Mathematics*, vol. 16, n° 2, 1966 :285-296.

[9] FENG LIN, « Opacity of discrete event systems and its applications », *Automatica*, vol. 47, n° 3, 2011 :496-500.

[10] CHRISTOPHE REUTENAUER, « Aspects mathématiques des réseaux de Petri », *Masson, Paris*, 1988.

[11] S. ABITEBOUL AND P. BOURHIS AND A. GALLAND AND B. MARINOIU, « The AXML Artifact Model, *In Proc. 16th Intl. Symp. on Temporal Representation and Reasoning (TIME)*, 2009.

[12] J. CLARK AND M. MURATA, « RELAX NG Specification, *http ://www.oasis-open.org/committees/relax-ng/spec-20011203.html*, 2001.

[13] MARGUS VEANES, « On Computational Complexity of Basic Decision Problems of Finite Tree Automata, *Uppsala University, Computing Science Department, January, 133, UPMAIL Technical Report, http ://research.microsoft.com/apps/pubs/default.aspx ?id=78382*, 1997.

[12] DOMINIC HUGHES, VITALY SHMATIKOV, « Information Hiding, Anonymity and Privacy : a Modular Approach », *Journal of Computer Security*, vol. 12, n° 1, 2004 :3-36.

[13] H. HOSOYA AND J. VOUILLON AND B. C. PIERCE, « Regular expression types for XML, *ACM Trans. Program. Lang. Syst.*, vol. 27(1), n° 46 ?90, 2005.

[14] CHRISTOPHE REUTANAUER, « Aspects Mathématiques des réseaux de Pétri, *études et recherches en informatique(eri), MASSON, Chaptire 3*

[15] B. BOUCHOU AND D. DUARTE AND M. HALFELD FERRARI ALVES AND D. LAURENT AND AND M. A. MUSICANTE, « Schema evolution for XML : A consistency-preserving approach, *Mathematical Foundations of Computer Science*, vol. 3153 of Springer Lecture Notes in Computer Science, 876–888, 2004

[16] E. BADOUEL AND M.A. BEDNARCZYK AND A.M. BORZYSZKOWSKI AND B. CAILLAUD AND PH. DARONDEAU, « Concurrent Secrets, *Discrete Event Dynamic Systems*, 17,4, 2007,425–446.

[17] B. FINANCE AND S. MEDJDOUB AND P. PUCHERAL, « The Case for Access Control on XML Relationships , *INRIA internal*, report 5446, 2005.